Kasitphoom Thowongs

```python
"""Increasing Subsequence"""
memorization = {}

"""Increasing recursive function to find increasing sequences"""
def increasing(k):

    if k in memorization:
        return memorization[k]

    if k == 0:
        return 1

    max_length = 0
    for i in range(k):
        if sample[i] ≤ sample[k]:
            val = increasing(i)
            if max_length < val:
                max_length = val
    memorization[k] = 1 + max_length
    return memorization[k]

def lis():
    max_length = 0
    for i in range(len(sample)):
        val = increasing(i)
        if val > max_length:
            max_length = val
    return max_length

sample = list(map(int, input().split()))
print(lis())
```

```python
"""Double, Triple, and Increment"""
memo = {}
def increment(x):
    if x in memo:
        return memo[x]
    if x == 1:
        memo[1] = 0
        return memo[1]

    if x % 3 == 0:
        memo[x] = min(increment(x / 3), increment(x - 1)) + 1
        return memo[x]
    if x % 2 == 0:
        memo[x] = min(increment(x / 2), increment(x - 1)) + 1
        return memo[x]
    if x - 1 > 0:
        memo[x] = increment(x - 1) + 1
        return memo[x]

print(increment(int(input())))
```

```python
"""Two Item finding"""
"""Function of finding sum of two numbers"""
def opt():
    target = int(input())
    arr = list(map(int, input().split()))

    arr = sorted(arr)
    low = 0
    high = len(arr) - 1

    while low ≠ high:
        calc = arr[low] + arr[high]
        if calc == target:
            return "Yes"
        elif calc < target:
            low += 1
        else:
            high -= 1

    return "No"

print(opt())
```

```python
"""Finding subset sum problem"""
mem = {}

"""Finding that is there a target in the set"""
def check(index, target):
    if (index, target) in mem:
        return mem[(index, target)]

    if index == 0:
        mem[(index, target)] = (target == 0)
        return (target == 0)

    if target < inputSet[index - 1]:
        mem[(index, target)] = check(index - 1, target)
        return mem[(index, target)]
    else:
        mem[(index, target)] = check(index - 1, target) or check(index - 1, target - inputSet[index - 1])
        return mem[(index, target)]

inputSet = list(map(int, input().split()))
target = int(input())
print(check(len(inputSet), target))
```

```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]

        merge_sort(L)
        merge_sort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

    return arr
```

```python
def quick_sort(arr):
    if len(arr) ≤ 1:
        return arr
    else:
        pivot = arr[0]
        less = [x for x in arr[1:] if x ≤ pivot]
        greater = [x for x in arr[1:] if x > pivot]
        return quick_sort(less) + [pivot] + quick_sort(greater)
```

```python
def largest_x(y):

    left = 0
    right = y

    while left ≤ right:
        mid = (left + right) // 2
        if mid * mid + mid ≤ y:
            left = mid + 1
        else:
            right = mid - 1

    return right
```

```python
list_numbers = list(map(int, input().split()))

def picking_number(list_num):
    if len(list_num) == 0:
        return True
    else:
        return picking_numbers(list_num[1:], list_num[0]) or picking_numbers(list_num[:-1], list_num[-1])

def picking_numbers(list_num, picked_num):
    if len(list_num) == 0:
        return True
    else:
        f_h = False
        s_h = False
        if abs(list_num[0] - picked_num) ≤ 9:
            f_h = picking_numbers(list_num[1:], list_num[0])
        if abs(list_num[-1] - picked_num) ≤ 9:
            s_h = picking_numbers(list_num[:-1], list_num[-1])
        return f_h or s_h
```

```python
def binary_search_approximation(arr, l, r, x):
    if r ≥ l:
        mid = l + (r - l) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] > x:
            return binary_search_approximation(arr, l, mid - 1, x)
        else:
            return binary_search_approximation(arr, mid + 1, r, x)
    else:
        return r
```

```python
def cutting(length, cuts):
    memo = {}

    def opt(l, r):
        if r - l == 1:
            return 0
        if (l, r) in memo:
            return memo[(l, r)]
        res = float("inf")
        for c in cuts:
            if l < c < r:
                res = min(res, r - l + opt(l, c) + opt(c, r))
        if res == float("inf"):
            res = 0
        memo[(l, r)] = res
        return res

    return opt(0, length)
```

Closest Pair of Points (Improved)

CLOSEST-PAIR(Px: list of points sorted in x, Py: list of points sorted in y)

Find vertical line L such that half the points are on each side of the line.	$\Theta(1)$
PxLeft= Px with all the points to the right of L removed	$\Theta(n)$
PxRight = Px with all the points to the left of L removed	$\Theta(n)$
PyLeft = Py with all the points to the right of L removed	$\Theta(n)$
PyRight = Py with all the points to the left of L removed	$\Theta(n)$
δ1 ← CLOSEST-PAIR(PxLeft, PyLeft).	$C'(n/2)$
δ2 ← CLOSEST-PAIR(PxRight, PyRight).	$C'(n/2)$
δ ← min { δ1 , δ2 }.	$\Theta(1)$
Delete all points further than δ from line L.	$\Theta(n)$
~~Sort remaining points by y-coordinate.~~	
Scan points in y-order and compare distance between each point and next 7 neighbors. If any of these distances is less than δ, update δ.	$\Theta(n)$
RETURN δ.	$\Theta(1)$

Cutting Stick

```
opt_profit[1…N] = Array of length n
opt_cuts[1…N] = Array of length n
opt_profit[1] = P[1]
opt_cuts[1] = []

for n = 2 to N:
  max_profit = P[n]
  max_cuts = []
  for k from 1 to n-1
    if max_profit < P[k]+opt_profit[n-k]-C
       max_profit = P[k]+opt_profit[n-k]-C
       max_cuts = [k] + opt_cuts[n-k]
  opt_profit[n] = max_profit
  opt_cuts[n] = max_cuts

print(opt_profit[N], opt_cuts[N])
```

29

Max 1D Range Sum – Kadane's Algorithm

- There is an O(n) algorithm to solve the Max 1D Range Sum problem. The algorithm described below is attributed to Jay Kadane.

```
// Kadane's Algorithm
sum = 0
max_sum = 0

for i = 1 … n
  sum += x[i]
  max_sum = max(max_sum, sum)
  if(sum < 0) sum = 0

return max_sum
```

- Try running the above algorithm on the array x =

  4, -5, 4, -3, 4, 4, -4, 4, -5

42

Max 2D Range Sum

- Given an n x n table of integers, find a pair (a, b, c, d) of locations in the table which maximizes the partial sum, i.e.

  sum(a, b, c, d) ≥ sum(x, y, x', y')

  for all indices w, x, y, z where x ≤ x' and y ≤ y'

- A straightforward implementation:

```
a = b = c = d = 1
max = x[a,b]

for i1 = 1 … n
  for j1 = 1 … n
    for i2 = i1 … n
      for j2 = j1 … n
        if(sum(i1,j1,i2,j2) > max)
          max = sum(i1,j1,i2,j2)
          a = i1; b = j1
          c = i2; d = j2
return (a,b,c,d)
```

51

Celebrity Problem

- **Attempt 4:** An improved version of Attempt 2.

```
function celeb_dq(S)
    if |S| == 1 then return the (only) person in S
    else
        Pick two people P1 and P2 from S.
        if P1 knows P2 then P'=P1 else P'=P2
        S' = S - {P'}
        C' = celeb_dq(S')
        if C' ≠ None then
            if P' knows C' and C' does not know P
            then return C' else return None
        else
            return None
```